BETA BASIC NEWSLETTER No. 5

Let's start as usual with corrections to the last Newsletter.

PROC rotate

Unfortunately the printers lost the last two lines from page 6. (From now on, I am supposed to check that their photographs match my originals and have not been wrongly trimmed.) My own contribution was to make a RESTORE statement incorrect by altering the line numbering slightly after PROC rotate had been converted to a Tasword text file. The missing lines are essential for poking the machine code for PROC rotate into place, and PROC rotate is needed as a sub-procedure for PROC dump in the same issue. Sorry about the mix-up - anyway, here is the corrected bottom of page 6:

```
70   IF PEEK cd<>33 THEN
        RESTORE 120
        FOR n=cd TO cd+37
           READ a
           POKE n,a
     NEXT n
```

Some readers managed to get everything working correctly despite the problems, and PROC dump was particularly appreciated. It can be modified to work with printers that require a non-Epson bit-image data arrangement. Larry Wooden (Clacton-on-Sea, Essex) got it working with his Apple Imagewriter by using rotates and changing the FOR in line 40 to FOR b=2 TO 9. PROC dump will work with the OPUS disc interface printer port to provide a screen dump facility.

POUND=HASH

The pound sign in line 50 of the procedure on page 3 of Newsletter 4 caused confusion in at least one reader. I simply forgot to change the printer to a non-UK alphabet; thus "#" (hash) came out as a pound sign. You'll often see this in listings in magazines.

A NOTE ON "TURTLE"

A minor problem with early versions of Beta Basic was that RENUM did not give a "Failed" message as it should on encountering LIST *expression*. This has been corrected, but the cure has the unforseen side effect of making TURTLE's TO statement mess up the display with a "Failed" message when it renumbers line 9975. This line is PROC LISTP - if you do not want to use this procedure, but want to avoid the TO problem, simply delete the line. (It is some *small* consolation to me to know that bug fixes to complex, programs are renowned for causing almost as many problems as they correct!)

****

LLISTINGS

Ettrick Thomson pointed out that even using the POKE to 57500 (given in No 4, p8) to control characters per line in printouts, the first line printed doesn't always do what is expected. This is because the value at 57500 is increased by one and copied to a printer position variable (57545) only when a carriage return is sent to the printer. So, do an LPRINT after POKEinq 57500, or POKE 57545 with whatever you used for 57500, plus 1.

****
128K VERSION OF PETA BASIC

I am feeling rather more positive about the 1228 that I was when I wrote the last Newsletter. Then, I was irritated at the needless inflexibility of the memory paging and the difficulty of hiding BB in the extra memory. Now, I am still irritated but I don't think it is such a handicap. The version I am working on now will still only give 22K or so for a Basic, program - but 22K is a *long program* and there would be no problem at all if you could MERGE new parts of the program from RAMdisc. But wait! What if you have a data-crunching program that takes 15K? You could easily run out of space for variables - almost always because of large arrays. If you could keep arrays on the RAMdisc and access them from the Basic program, the problem would be solved. I intend to allow individual arrays up to 64K in length. A data-crunching program could have 70 or 80K of variables - much better than would normally be possible without Beta Basic.

I should stress that this doesn't work yet - but I'm pretty sure it will. A 128K version of Beta Basic should be ready in time to give details in the next Newsletter. I know many BB users have held off buying a 128 until they are sure that a BB version able to use the extra memory is coming - hence this advance notice The 128K Spectrum is selling well and Betasoft will support it.

Both the Spectrum Plus and the Plus 2 have a good video signal should give a much improved picture on a monitor or monitor TV. Once you have used one you will find it hard to go back to a conventional TV. I use a Ferguson TV with RGB input which I would recommend. It is just as good as a "real" monitor at the resolution available on a Spectrum. Incidentally, I think that the Plus 2 keyboard is quite good.

****
PROC short - truncating strings from arrays

Strings in arrays are always padded with spaces to make them the same length, which can be inconvenient. PROC short (from Alain Vezes, Albi, France) strips off the trailing spaces from a string supplied as the first parameter. The result goes into a string specified by the second parameter. The example shows that PROC short still works with the special case of a string that is all spaces thanks to the EXIT IF L=O. (One could also use LOOP UNTIL L=0.)

```
10   DIM as(10,10)
     LET a$(1)=”testing"
        short a$(1),b$
     PRINT b$,LEN b$
        short a$(2),b$
     PRINT b$,LEN b$

100  DEF PROC short x$, REF y$
        LOCAL L
        LET=LEN x$
        DO WHILE x$(L)=” “
           LET L=L-1
        EXIT IF L=0
        LOOP
        LET y$=x$( TO L)
     END PROC
```

****
SIMPLE MENUS

Ivan Klemvig (Chirk, Clwyd) recently asked about using a "menu" to select from a set of procedures. Below is a possible basis for a simple menu system. The DO-loop prints the available options (you would almost certainly want to do this more prettily!) and GETS a number from the keyboard. This causes one of the (silly) procedures after ON to be selected. After that procedure has finished, the main loop will be re-entered at line 110, and the menu will be printed again. The only frill I have included is an exit option using RROC leave. This PROC sets a flag (yn) so that the higher-level PROC smenu leaves the DO-loop and finishes. The alternative of just putting a "STOP" in PROC leave is simpler but it leaves the PROC/GOSUB stack and local variables in a "messy" state.

```
10   smenu

100  DEF PROC smenu
        LOCAL k,yn
        LET yn=O
        DO
           CLS
           PRINT "1. Noise"`"2. Number"`"3. Exit"
           GET k
           ON k
              noise
              rannum
              leave yn
110     LOOP UNTIL yn=1
     END PROC

120  DEF PROC noise
        BEEP 1,1
     END PROC

130  DEF PROC rannum
        CLS
        PRINT RNDM(10)
        PAUSE 100
     END PROC

140  DEF PROC leave REF abort
        PRINT "bye!"
        LET abort=l
     END PROC
```

****
HORIZONTAL MENUS

The following contribution from P. Field (Halsham, N. Hurnbs.) is a more complex variant on the menus theme; I have modified it to avoid the need for pre-defining particular windows and variables. The procedure takes up to 5 options as a DATA list and prints them on the top screen line. Pressing any key except ENTER will move the "current selection" by one; pressing ENTER will cause the procedure to finish with global variable "opt" holding the final selection.

Notes:

Variable names have been "re-used" b$ is DIMed as 10 to make the READ selections all the same length, and b$ is also used with GET - hence the reference to b$(1).

In line 310 the quotes contain 10 spaces.

The "current selection" is indicated initially by printing in a contrasting colour, using OVER 1 and spaces so that the underlying text isn't erased. The colour alone is then moved using ROLL (attributes) to change the highlighted option. (The variable n changes too.) If you try to move too far right, the highlight is ROLLed back to the left margin again.

```
200  menu M-drive,Forget, Frinter,Exit
     PRINT   opt

300  DEF PROC menu DATA
        LOCAL b$,n
        LET opt=0
        PRINT CSIZE 4,8;AT 0,0;STRING$(64," ")
        PRINT AT 0,0;
        DIM b$(10)
        DO
           READ LINE b$
           PRINT CSIZE 4,8;b$;"   ";
           LET opt=opt+1
        LOOP UNTIL ITEM()=0 OR opt=5
310     PRINT CSIZE 4,8;AT 0,0; OVER 1; PAPER 2; INK 7;"
         ": REM 10 spaces

320     LET n=1
        DO
           GET b$
        EXIT IF b$(1)=CHR$ 13
           LET n=n+1
           IF n<=opt THEN
              ROLL 4,48;0,175;32,8
           ELSE ROLL 1,(opt-1)*48;0,175;Z2,8
              LET n=1
330     LOOP
        LET opt=n
     END PROC
```

****
BIGGER DUMPS

PROC dump in issue 4 produces a screen copy that is about 8 cm. (3.1 inches) wide on an Epson printer. This is a little small for some purposes. It is easy to double the width of the dump - simply send each byte twice by changing the LPRINT a$(b); in line 40 to LPRINT aS(b);a$(b);. You must also alter line 30 so that 512 bit-images bytes are specified - the CHRS 1 will become CHR$ 2 Doubling the height is a bit harder. In some cases you use CSIZE 8,16 to double the on-screen height, or alternatively add the following statements between GET and rotate in line 40:

```
LET a$(9)=a$(5),a$(8)=a$(5),aS(7)=a$(4),a$(6)=a$(4)
LET a$(5)=a$(3),a$(4)=a$(3),a$(3)=a$(2)
```

Change the FOR r statement to: FOR r=0 TO 21.5 STEP 0.5

****

TURNING PROGRAMS INTO TASWORD TWO FILES

Jan Biemans (Zoetermeer, The Netherlands) asked for my method of transferring programs to the Tasword word processing program, so here it is, for Tasword Two. Due to inherent indolence and because I have a large number of old text files I haven't converted to Tasword Three yet - but see the next section.

First, load the program you want to transfer. Then OPEN a stream for output to a serial file - make sure it is a new name. Then LIST the program to the stream, and close it. e.g.

```
OPEN #5;"m";1;"test"
LIST #5
CLOSE #5
```

If you do this with BB resident the program will be output according to the current LIST FORMAT. For the next stage, BB must be present. LOAD the program below and run it. It will
ask for the input file name e.g. "test" and then the name for the output text file. The locations normally occupied by the first 100 lines of a Tasword file are cleared to spaces by the POKEd STRINGS - clear more space if you need to. The input file is loaded one string at a time and processed. (The carriage returns at the end of each program line, and any carriage returns caused by colons cause the division into strings.) Since the file contains tokens rather than ASCII, SHIFT$ is used to expand them. The result is printed, and poked into memory. The value 32001 is used in line 20 to give an left indent of one space, but you could use 32000, 32002 or other values. Finally the poked memory area is saved and can be loaded back like any other Tasword file. The main flaw in the program is that long program lines are not indented from the left margin when they exceed 64 characters and wrap round to the next line, but this is really only a nuisance with long DATA statements.

```
10   CLEAR 31999
     INPUT "Input file? ";i$
     INPUT "Output file? ";o$
     POKE 32000,STRING$(6400," ")
     CLOSE #6
     OPEN #6;"m";1;i$
     LET d=0
20   DO
        INPUT #6; LINE a$
        LET a$=SHIFT$(7,a$)
        FOR c=1 TO LEN a$
           PRINT a$(c);
           POKE 32001+c+d,CODE a$(c)
        NEXT c
30      PRINT
        LET d=d+64+64*INT (LEN a$/64)
     LOOP UNTIL EOF(6)
     SAVE 1,o$ CODE 32000,d
```

****

TURNING PROGRAMS INTO TASWORD THREE FILES

One of the disadvantages of Tasword Two is that a text file of e.g. 200 lines always takes 200*64 bytes, even if most of the lines are empty or very short. Tasword Three uses a different system in which short lines of text take less space, using CHR$ 13+CHR$ 10 as a terminator. This makes the conversion of programs into text files rather easier, as both input and output files are essentially collections of strings of irregular length. The main task is expansion of keyword tokens to ASCII form. The solution is similar to the program on page 80 of the BB manual This contribution from M.J. Smith (Rossendale, Lancs.) can be MERGEd with the program in memory. It will create a data file of all preceding program lines on drive 1 with the name PROGLIST, which can be loaded or merged as a Tasword Three document. The LIST FORMAT can be adjusted as required.

```
9300 DEF PRROC IMPORT
9310    LIST FORMAT 2
9820    OPEN #5; "M"; 1; "TEMP"
        LIST #5; TO 9799
        CLOSE #5
9830    OPEN #5;"M";1;"TEMP"
        LET AS=””
9840    DO UNTIL EOF(5)
9850       INPUT #5; LINE Z$
9860       LET A$=A$+SHIFT$(7,Z$)+CHR$ 13+CHR$ 10
5870    LOOP
9880    CLOSE #5
9890    ERASE "TEMP"
        ERASE "PROGLIST"
        OPEN #5;"M";1;”PROGLIST"
        PRINT #5;A$
        CLOSE #5
9990 END PROC
```

****

PASSING PARAMETERS BY REFERENCE - A MISUSE

Passing parameters by reference is one of Beta Basic's more subtle features. Although powerful, it can also be confusing. One thing you should *not* he doing is passing a FOR-NEXT loop variable by reference. YOU only, need to use REF to specify a variable which will be altered so that the procedure can return a result. You should not be altering a FOR-NEXT variable within the loop in any case, but if you pass such a variable to a procedure as a REF parameter, you will cause a problem. The parameter will be turned into a normal numeric variable, except in the special case when the formal and the actual parameters are the same Normal single-letter numeric variables generally act the same as FOR-NEXT variables, but they don't satisfy NEXT. You will get a "NEXT without FOR" error. Here is an example:

```
100  FOR F=1 TO 10
        test n
        PRINT n
     NEXT n

200  DEF PROC test REF x
        LET x=-5
     END PROC
```

Passing a FOR-NEXT variable *without* using REF is quite legitimate, of course - try removing REF from the example.

****
MORE ON FAT CHARACTERS

Stan Flynn (Stafford) wrote to me after reading Newsletter 4. He suggested the much simpler and faster method of "fattening" characters given below. Text is plotted twice at a desired x and y coordinate, with the start position being moved by one pixel for the second plot. OVER 2 makes this add to the first plot.

```
10   FATCHAR "THIS IS AN EXAMPLE",55,87

100  DEF PROC FATCHAR a$,x,y
       FOR f=x TO x+1
         PLOT OVER 2;f,y;a$
       NEXT f
     END PROC
```

Later, Pauli Lindgren (Helsinki, Finland) sent a more elaborate version of the same idea that uses character coordinates instead of pixel coordinates. The default position is the previous print position (obtained by PEEKing the relevant system variables), so usually you don't need to specify any coordinates at all. After the text is written, the print position is moved to the end of the text. You can use PRINT to move the print position down to the next line.

```
10   PRINT "normal"
20   bold "HOLD CAPITALS"
30   PRINT
40   bold "and bold lower case"

100  DEF PROC bold t$,rew,col
       DEFAULT cot=33-PEEK 23688,row=24-PEEK 23689
       LOCAL x,y
       LET x=cot*8,y=175-row*8
       PLOT x,y;t$
       PLOT OVER 2;x+1,y;t$
       PRINT AT row,col+LEN t$;
     END PROC
```

****
ITALIC PRINTING

Pauli Lindgren also sent a PROC to print italics which works similarly to the previous PROC bold. I have modified the FOR-NEXT loop that "smears" the text sideways - you might like to play with this too, in order to vary the effect.

```
200  DEF PROC italics t$,row,col
       DEFAULT col=33-PEEK 23688,row=24-PEEK 23689
       LOCAL x,y,n
       LET x=col*8,y=175-row*8
       PLOT OVER 2;x,y;t$
       FOR n=1 TO 7 STEP 2
         SCROLL 8;x,y;LEN t$+1,n
       NEXT n
       PRINT AT row,col+LEN t$;
     END PROC
```

Pauli adds that you can get bold italics with something like:

```
BOLD "text",10.10: ITALICS "      ",10,10
```

****
3D HIGHLIGHTING

Once prodded by Stan Flynn, I realised that over-plotting in various waves could produce some interesting effects. I expect readers will have their own ideas - but here is one of mine:

```
10   hilite "BETA",0,87

200  DEF PROC hilite a$,x,y
       FOR n=0 TO 3
         PLOT CSIZE 32; OVER 1;x-n,y-n;a$
       NEXT n
         PLOT CSIZE 32; OVER 2;x-4,y-4;a$
     END PROC
```

****
LOGO-STYLE REPEAT STRUCTURE

My first thought on seeing this contribution from Pauli Lindgren (Helsinki) was that it couldn't work. My second was that if the real structured programming fans catch him, they will have him shot. I took some time to understand the program logic (there are oddities to do with keeping the PROC/GOSUB stack in balance) and I suggest you don't bother! However, the structure works very well (it is better than the one in TURTLE) and apparently has no nasty side effects. Deviousness seems to be effective!

Pauli writes: "REPEAT n repeats n times all statements inside the block. Each block ends with an END_ REPEAT statement. Loops can be freely nested and used within procedures. The only restriction is that each REPEAT and END_ REPEAT must be on a separate line. Note also, that the loop variable n of the innermost loop is visible to the main program within the loop."

```
10   DEC PROC repeat n
       LOCAL retline
       POP retline
       GO SUB 20
20     FOR n=1 TO n
         POP
         GO TO retline+l
30     END PROC

40     DEF PROC end_repeat
         GO SUB 50
       NEXT n
50   END PROC

100  repeat  5
110    repeat 10
120      PRINT "*";
130    end_repeat
140    PRINT
150  end repeat
```

The demonstration lines 100-150 have been manually indented. Incidentally, I recently told to a reader that DEF PROC and END PROC should appear next to the left-hand margin to a LLISTinq or something was wrong with the program. PROCs REPEAT and END_REPEAT are the exception that proves the rule!

****
PRINTING ERROR Messages - AND A METHOD OF HANDLING PACKED TEXT

A reader asked if I could add an error message string facility to Beta Basic. This would be useful with ON ERROR, as you could print an error message derived from the variable ERROR without having to make up a whole set yourself. All the normal error messages are of course stored in the ROM, and BB's are part of the program, so I wondered how easy it would be to use them. Both sets use a very common (in machine code) method of running messages together, with 128 being added to the CODE of the last character in each massage. The messages are accessed by counting past a number of such end markers till you reach the message you want. Unfortunately, when I tried this in Basic, it took several seconds to scan through even 20 messages. In machine code, a very simple routine will do the trick, as shown by PROC locate. The procedure looks through memory starting at *ad* for string *i* (in the range 0-255) and returns it in the string you specify. The machine code routine is POKEd into the printer buffer (but you could use another location) if it hasn’t been done already. It is used twice, once to find the start of the string and once to find the end. SHIFT$ is used to remove the "plus 128" marker from the last character. PROC errmes determines whether the error message wanted is a Spectrum Basic or a Beta Basic one, and sets p appropriately before calling PROC locate to find it.

```
10   FOR e=0 TO 33
        errmes e
     NEXT e

100  DEF PROC errmes er
        LOCAL p
        PRINT er;
        LET p=5009
        IF er>=28 THEN
           LET er=er-28,p=62042
110     locate p,er,a$
120     PRINT TAB 4;a$
130  END PROC

200  DEF PROC locate ad,i, REF a$
210     LOCCAL s,n,a
220     LET s=23296
        IF PEEK s<>6 THEN
           FOR n=s TO s+14
              READ a
              POKE n,a
           NEXT n
           DATA 6,0,33,0,0,126,35,23,48,251,16,249,68,77,201
230     POKE s+l,i+l
        DFOKE s+3,ad
240     LET ad=USR s
        POKE s+1,1
        DPOKE s+3,ad
        LET a$=SHIFT$(5,MEMORY$()(ad TO USR s-1))
250  END PROC
```

You might want to modify PROC locate to look through your own strings, since you can run up to 256 strings together into one giant string, without the wastage of space that occurs in arrays. Chance the *ad* in line 200 to e.g. b$, and the *ad* in line 230 to LENGTH(0,"b$"). The big string you make up should have a dummy first character with a CODE of 128 or more. Call the PROC with the name of this string as the first parameter.

****

TWO SPECTRUM BUSS .... AND A FEATURE

Michael Charlton (Pontefract, W. Yorks.) noticed while altering PROC axes (issue 3) that:

```
     PLOT x,y;STR$ x
```

often leads to strange results, but the form below always works:

```
     LET a$=STR$ x: PLOT x,y;a$
```

This is due to a bug in the part of the Spectrum that deals with printing decimal numbers. If the number is between -1 and 1, an extra number is left on the stack used during expression evaluation. This causes problems for the STR$ routine which "prints" numbers to a temporary "workspace". Thus:

```
     PRINT 2"+STR& O.5
```

gives 0.5, not 20.5 as it should. When used with BB's PLOT string command, the effect is to make the plot occur at the wrong place, as in:

```
     PLOT 100,100;STR$ O.5
```

Unfortunately this is hard to cure without a ROM change! You can program round the bug by pre-assignment of the string as above.

Michael also noticed while saving strings that they would only load back if he had dimensioned them first. Actually, the SAVE "name" DATA a$() command is only supposed to work with arrays, and any attempt to save an ordinary string should produce an error message. Due to a mistake in the ROM, this doesn't happen, and saving is allowed - but the strings cannot be loaded back!

THE FEATURE

Rod Macaulay noticed while writing educational programs in Seta Basic that a comparison like: IF z=y*10 THEN ... can give the wrong result. Another example of the same phenomenon is:

```
     PRINT 1.2*10-12
```

The expected result is zero, but instead a small number is printed in exponent form. The problem is seen on many computers, and it arises because floating-point numbers cannot always be stored with complete accuracy in the binary format generally used. (What is printed on the screen is a rounded version of the internal format - otherwise PRINT 1.2*10 would be seen as 12.0000000037 and the arithmetic failure would be a bit less confusing.) The solution is to round decimal numbers somehow before doing any comparisons. With Beta Basic we could use:

```
     DEF FN b(x)=VAL USING$("#######.######",x)
```

which will round a number to six decimal places. (Change the number of hashes after the decimal to modify the rounding precision.) Six decimal places of any stored number should De correct, apart from cumulative errors in calculations. so:

```
     PRINT FN b(1.2*10)-FN b(12)  gives zero.
```

****
OLD-STYLE NUMERALS

This contribution is from Ettrick Thomson (Aldeburgh, Suffolk). The method of copying the normal character set from RCM to RAM for modification should be of interest to anyone who wants to customise all or part of the character set; PROC chardes from issue 4 could be used for this purpose with very minor changes. Here's Ettrick's explanation:

"The procedure 'OS_Num' replaces the 0,1,2,...,9 of the Spectrum Character Set by old-style versions. These have ascenders and descenders which, as with lower-case letters, help in distinguishing pairs of characters that resemble one another in the normal set, such as 'B/8' or 'S/5'."

"These old-style numerals have long been standard in UK and European mathematical tables and (sometimes in modified form) in UK and European typewriters. The USA, however, does not use them and it is presumably their influence that has led to their disappearance from computers."

"Line 130 is an optional extra. It goes on, after changing the numerals, to "fatten" the new character set (Newsletter No 4, p.2), adding about 15 sets to the running time of a few seconds without it; the copyright character, however, is excluded, since it would be incompletely fattened and would be lop-sided. These fattened characters are highly recommended on the Alphacom printer."

"It is worth pointing out that Line 100 copies the normal character set to a new location above a suitably reduced RAMTOP, and makes the System Variable CHARS point to this new set. This could be a preliminary to any partial change of the character set. The Beta Basic NEW (unlike the straight Spectrum NEW) does not affect CHARS."

The listing is a screen dump showing the new characters in use.

```
100  DEF PPOC OS_num
        LOCAL s,i,c
        CLEAR 344
        CLEAR 384
        LET  s=DPEEK (23730)+2
        DO UNTIL PEEK (s-i)=0
           LET s=s+3+DPEEK(s)
        LOOP
        POKE s,MEMORY$() (15616 TO 16383)
110     RESTORE 120
        FOR i=s+128 TO s+207
           READ c
           POKE i,c
        NEXT i
        RESTORE
120     DATA 0,0,56,76,84,100,56,
        0,0,0,48,16,16,16,56,0,0,
        0,56,68,24,32,124,0,0,0,5
        6,68,8,4,68,56,0,0,8,24,4
        0,72,252,8,0,0,120,64,120
        ,4,68,50,0,16,32,120,68,6
        8,56,0,0,0,124,4,8,16,32,
        64,0,65,68,56,68,68,56,0,
        0,0,56,68,68,56,8,16
130     FOR i=s+8 TO s+759
           POKE i,OR(PEEK i,PEEK i
           /2)
        NEXT i
140  END PROC
```

****

BETA BASIC SYSTEM VARIABLES

Here is a long-promised item - the main block of BB system variables. (There are a few more scattered about, some of them previously documented.) Some variables are used for several or many different purposes. Note that variables may be 1 or 2 bytes long. Happy hacking!

57344	Multiple uses.
57345	Multiple uses.
57346	Multiple uses.
57348	FILLed area/temporary store for output routine address.
57350	Temporary store for attributes used by PLOT a string.
57351	Temporary store for INVERSE/OVER used by PLOT a string.
57352	Temporary store for address of the output routine.
57354	Statement counter used in syntax check.
57355	Flip-flop used by TRACE.
57356	START used by commands which allow start TO end
57357	END as above. Also used as a pointer during PROC.
57360	Spare - I think!
57361	Output flags. Bit 7 is 1 if CSIZE is non-0. Bit 0 shows if a GET block has attribute information or not.
57362	OVER 2 status - temporary.
57363	OVER 2 status - permanent.
57364	current co-ordinates for non-zero CSIZE.
57366	right limit of current WINDOW
57367	left limit
57368	top limit
57369	bottom limit
57370	Permanent character width
57371	Permanent character height
57372	Temporary character width
57373	Temporary character height
57374	Height magnification of characters
57375	Counter used during character height magnification.
57376	Temporary WINDOW (0, or WINDOW no. +128)
57377	Indentation counters used by LIST FORMAT.
57381	LIST FORMAT flag. 1=indented, 0=normal
57382	Line numbering flag. Non-0=no line numbers, 0=line numbers
57383	KEYWORDS mode 2/3/4
57384	Screen height in character lines.
57385	Cursor flag used in fast cursor movement.
57386	Flag showing scrolling was needed to show the current line
57387	Previous scroll count - used in fast cursor movement.
57388	Line with cursor
57389	Number of screen lines in a program line - see below.
57390	Number of characters in addition to lines. Used as counters during fast cursor movement.
57391	Characters per line.
57392	Flag used by output routine during indentation.
57393	EDIT flag. 0=EDIT, 255=INPUT
57394	AUTO flag
57795	DEFAULT drive or station number.
57396	DEFAULT device letter (m/t/n)
57397	LET flag. 0=LET, 255=DEFAULT (variable)
57398	SAVE etc. variable. 0=no slicer, 255=slicer, 228=DATA
57399	Flag used by REF
57400	Store for error stack pointer
57402	AUT0 step value/REF data
57404	Counters used by ROLL/SCROLL to keep attributes in synch.
57406	ALTER reference/REF variable
57407	Permanent WINDOW

****

DISC SYSTEMS

Armin Kausler (Boblingen, West Germany) Writes:

" I use my Spectrum with a Timex Floppy System. There is no other way to send or receive data from the floppy than to enter:

```
   RAND USR 59904: CAT * (or LOAD #, SAVE # etc.): RAND USR 58419
```

But I cannot use this sequence of commands inside PROCs, LOOPs, etc. Can you tell me how to switch BB 3.0 on/off without this disadvantage? "

Sometimes users require to turn BB off briefly while some add-on is used. RAND USR 59904 turns BB off, but it also clears the stack used by PROCs, LOOPS, that when BB is turned back on the PROC or LOOP cannot continue correctly. If you use instead:

```
   DPOKE DPEEK(23613),4867: DPOKE DPEEK(23613)-2,7030
```

Beta Basic will be turned off but the stack will be left intact. BB can be turned back on by RAND USR 58419.

Peter Safranek (Ashford, Middx.) and Albert Kleyn (Romsev, Hants - he is a product manager for IBM UK) have pointed out that the most recent versions of the Technology Research Beta Disc Interface (version 5.01) live with Beta Basic less happily than earlier versions did. The problem is avoided by switching Beta Basic off before using any disc commands. The problem seems to be related to BB'S use of Interrupt Mode 2. This can be controlled by:

```
      RANDOMIZE USR 63243 (Set normal Interrupt Mode 1)
      RANDOMIZE USR 61369 (Set Interrupt Mode 2)
```

The advantage of this method is that you can switch permanently to Interrupt Mode 1 and Beta Basic will still work, apart from the CLOCK and the improved BREAK. It may only apply to the Beta Disc Interface, though.

Peter sent a listing of a program that loads Beta Basic and sets up useful DEF KEYS for controlling TR DOS (version 5) before deleting itself. I will only include enough of it here to give the general idea. Note the use of one of the DEF KEYs to give a HELP menu for the others.

```
      2    CLEAR rt: RANDOMIZE USR 15619: REM: LOAD "beta 3.0"CODE
      3    RANDOMIZE USR 59419
      30   DEF KEY "c": RANDOMIZE USR 59904: RANDOMIZE USR 15619:
           REM : CAT
      40   DEF KEY "L": INPUT "Basic prog name? ";a$: RANDOMIZE
           USR 59904: RANDOMIZE USR 15619: REM: LOAD a$
      50   DEF KEY "k": INPUT "Code prog name? ";a$: RANDOMIZE
           USR 59904: RANDOMIZE USR 15619: REM: LOAD a$CODE
      90   DEF KEY "h": CLS: PRINT "BETA BASIC 3.0 - DOS DEF KEYS:”
           '''"c CAT" '' "l LOAD" '' "h help"
      110  DELETE TO
```

Since I mentioned in the last Newsletter that I had some OPUS contributions I wasn't going to publish, I have had many letters requesting publication. The contributions have been mainly improved versions of CAT, the best one being by Alexander Stols of Molenhoek, Netherlands. (I hadn't realised that OPUS allows any disc file to be OPENed as a serial file for input - this means you can read the header information on type, start line, variables length, etc. using successive INKEY$'s from the file.) I also have a routine, adapted from an Interface 1 version, that PRINTS to a string using stream #14. I think most OPUS users want a good random-access database, which should be quite easy, but I have not seen one yet. I am not sure that this Newsletter is the best forum for such things, either. John Wase (a Newsletter subscriber) writes an OPUS column for ZX COMPUTING and would welcome contributions. (*Apart from* improved CATS!)

****

PROC/GOSUB/DO-LOOP STACK AND LOCAL VARIABLES

If you press BREAK, or some other error occurs during a procedure, and you do not CONTINUE or later use RUN or CLEAR, return addresses and local variables can build up. To avoid this, enter END PROC after using BREAK. You can do this several times, until "Missing DEF PROC" shows that the data has been cleared; this might be necessary if procedures were nested. POP removes a return address from the stack used by GOSUBs, PROCs and DO-loops, but it does not delete local variables. Since even a procedure without parameters is assigned an empty storage space for local variables, you cannot "clear up" after a stopped procedure using POP alone. There is a way of deleting both a set of local variables and a return address while staying within a procedure, although it is inelegant. I have illustrated it below by modifying PROC end from issue 4 (lines 100-130 below) so that it leaves the stack and local variables clear. Lines 10 and 20 place two items on the stack; line 30 adds another and creates a local variable storage area, and finally PROC end2 is entered.

```
10   GO SUB 20
20   GO SUB 30
30   test

40   DEF FROC test
       end2
     END PROC

100  DEF PROC end2
110    IF DPEEK(23613)+3<DPEEK(23730) THEN
         GO SUB 130
         POP
         GO TO 110
120     GO TO 10000
130  END PROC
```

PROC end2 looks at two system variables, 23613 (ERR SP) and 23730 (RAMTOP). The PROC/GOSUB/DO-loop stack grows downwards from RAMTOP, lowering ERR SP. When this stack is empty, ERR SP is 3 less than RAMTOP. If the stack is not empty, the GO SUB 130 puts another item on the stack; the END PROC at line 130 erases a set of local variables and uses the stack address to return to the POP statement. (END PROC cannot distinguish between a GOSUB return address and a procedure return address.) POP removes a stack item and the loop will continue until the, stack is empty. It doesn't matter if the local variable sets are all erased before the return addresses "run out".

****

REMOVING COLOUR CONTROL CODES

Here is an interesting use of ALTER (a program reference) from Bo Leuf (Gothenburg, Sweden). It removes any INK, PAPER (first FOR j loop) FLASH, BRIGHT or INVERSE (second FOR j loop) control codes embedded in the program. Note the form of ALTER used in PROC strip to remove any instances of the string specified by a$.

```
1  DEF PROC decolor
      LOCAL i,j
      FOR j=16 TO 17
         FOR i=0 TO 9
            strip
         NEXT i
      NEXT j
      FOR j=18 TO 20
         FOR i=0 TO 1
            strip
         NEXT i
      NEXT j
   END PROC

2  DEF PROC strip
      LOCAL a$
      LET a$=CHR$ j+CHR$ i
      ALTER (a$) TO ""
   END PROC
```

****

SPECIAL OFFER

I have recently used HiSoft's Basic compiler and I like it very much. It successfully compiled a 20K Basic program which I had lying around which was full of array references and user defined functions - things which are not generally compilable. The result was about the same length but 20-30 times faster, which is pretty good. The compiler is fast, user-friendly, and can be copied to disc or Microdrive. It will compile Basic programs up to 30K long, or even more and the 128. It deals with all Spectrum Basic (bar a few very minor exceptions) and it allows mixed integer (for speed) and floating-point arithmetic.

It won't, of course, compile Beta Basic programs, but you can locate the compiled code anywhere, so you could compile all or part of a Spectrum Basic program and use it with BB by loading it below BB's code. You can use several different USR calls to enter at different places in the compiled code (calling different compiled subroutines, for example).

There have been a number of highly-hyped compilers on the market, but this is definitely the best I've seen. I have arranged a discount of £2.50 with HiSoft for BB Newsletter subscribers, which makes the price £ 22.45. If you send cheques for this amount, made out to HISOFT, to this address, we will certify them as coming from members and forward them to HISOFT, who will send you the program. (They send stuff fairly fast - i.e. by return of post - so even this circuitous method should give better service than you can expect from many companies.)

In case you are wondering, Betasoft makes no money at all out of this arrangement!

****
LAST WORD

In case you wonder what happened to "Readers' Letters", in this issue I have treated topics raised by readers, rather than quoting letters directly - it saves space. Still, there will probably be some next issue.

Please send some contributions - I've almost run out!

The Post Office

Many people suspect the Post Office of being unreliable; after all, it is pretty common to hear the phrase "Terribly sorry - it must have got lost in the post!" So how bad are they? Well, out of a fair number of items moving between Betasoft and companies „about one-third were apparently "lost". In one case we learned by other means that the item had never been sent; in another we were fortunate to get the item, since the road name was missing! Wrongly--typed addresses seem to be common.

Out of many thousands of items moving between Betasoft and mail-order customers in the U.K., the score is as follows:

Two newsletters not received - maybe the label fell off, or the too-small envelope disintegrated.

One copy of Beta Basic not received - the replacement was found tucked behind a door (stealable), apparently because the letter slot was a bit narrow.

One order for BB not received - the customer wrote something like "If by any chance you didn't get the order I sent recently that is a real pity because it was postal orders and I didn't keep the counter-foils and I'm an invalid and I haven't any money." We thought this was a bit thin.

We also sent one letter to the wrong address ourselves, by (somehow) combining two addresses on the word processor!

Three cheers for the British Post Office! If something is "lest in the post" the most likely explanation is that it was wrongly packed or labeled by the sender, and the next most likely explanation is that the item was never sent. It is much, much less likely that the Post Office actually lost it.

Outside the U.K., the picture is patchier. Some countries (West Germany, Norway, Sweden, Denmark, Holland) are very reliable. Others (Greece, Spain, Portugal, India) are rather less so. It is always a bad sign if the inhabitants of a country routinely send ordinary letters by registered post - they must know something!

BETASGFT, 92 OXFORD ROAD, MOSELEY, BIRMINGHAM, B13 9SQ, ENGLAND